DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
SYSTEMS ENGINEERING LABORATORY
THE UNIVERSITY OF MICHIGAN, ANN ARBOR

2up

SEMI-ANNUAL STATUS REPORT

(Covering the period from 1 January 1974 to 30 June 1974)

NASA RESEARCH GRANT NGR23-005-622

THEORY OF RELIABLE SYSTEMS

Principal Investigator

John F. Meyer

July 1974

TABLE OF CONTENTS

	Page
1. Objectives	1
2. Personnel	3
3. Technical Status	4
3.1 Reliability Analysis	6
3.1.1 Background	6
3.1.2 Computers with Faults	9
3.1.3 Tolerance Relations and Erroneous Computations	18
3.1.4 Reliability Measures	23
3.1.5 Topics for Further Investigation	31
3.2 On-line Diagnosis	34
3.2.1 Background	34
3.2.2 Recent Activity	37
3.2.3 Topics for Further Investigation	40
4. References	44

1. OBJECTIVES

The purpose of this research project is to refine the current notion of system reliability by identifying and investigating attributes of a system which are important to reliability considerations and to develop techniques which facilitate analysis of system reliability. Attributes selected for investigation included:

(a) Fault tolerance - the ability to maintain error-free input-output behavior in the presence of (temporary and/or permanent) faults in the system

(b) Diagnosability - the ability to detect and locate faults in the system

(c) Reconfigurability - the ability to reconfigure the system after the occurrence of a fault so as to realize the original behavior or some other (possibly less complex) behavior

with the following proposed objectives:

I. To determine, relative to the above attributes, properties of system structure that are conducive to a particular attribute. Structures so considered will range from state-transition functions at one extreme to hardware and software realizations at the other extreme.

II. To determine methods for obtaining reliable realizations of a given system behavior. In particular, one would like to obtain realizations which are fault tolerant (relative to the specified behavior) and

yet diagnosable (relative to some extended behavior).

III. To determine how properties of system behavior relate to the complexity of fault tolerant (diagnosable, reconfigurable) realizations. Once such relationships are discovered, the inherent fault tolerance (diagnosability, reconfigurability) of a given behavior could be measured by the minimum complexity of realizations possessing that reliability attribute.

IV. To determine methods for evaluating the reliability of a proposed or existing system as measured in terms of fault tolerance, diagnosability, reconfigurability, or combinations of these attributes. This includes the investigation of appropriate reliability measures, modeling techniques, and computational methods for determining, or at least estimating, system reliability.

Since the initiation of the grant, the above proposed objectives have been augmented to obtain a more definitive statement of what research should be accomplished to meet the needs of NASA and, in particular, the Langley Research Center. The following statements of these augmented objectives are due primarily to the constructive suggestions of NASA-Langley, with some subsequent modification in wording to conform more closely with our interpretation:

I. To develop formal concepts and establish mathematical results which can be used to precisely define measures of system utility, e.g.:

1. Measures of fault tolerance;

2. Measures of recoverability based on measures of detectability,

locatability and reconfigurability;

3. Measures of system availability with respect to different levels of system performance;

4. Measures of total system "worth" based on measures of performance worth and measures of performance availability.

II. To develop analytic and simulation methods for evaluating system utility measures.

III. To determine architectural characteristics of fault-tolerant systems that are amenable to fault detection and fault location.

IV. To investigate methods of on-line diagnosis that are applicable to specific subsystems of a fault-tolerant computing system, e. g. :

—given an arithmetic unit subject to a specified class of faults, design a detector that, with a specified allowable time delay, will detect any error produced by a fault.

V. To investigate methods of augmenting the structure of specific hardware or software subsystems in order to facilitate detector design and improve on-line diagnosability.

2. PERSONNEL

To meet the objectives stated in Section 1, it was estimated that the following technical effort would be required:

Principal Investigator
25 percent time, academic year
100 percent time, two months, summer

Research Assistants
1 at 50 percent time, academic year
2 at 25 percent time, academic year
3 at 100 percent time, summer

Programmer
25 percent time, fiscal year

During the period 1 January-30 June 1974 (referred to as the "reporting period") research personnel and their level of effort have been:

Principal Investigator
John F. Meyer
25 percent time, January - May
100 percent time, June

Research Assistants
David E. Frisque
20 percent time, January - April
100 percent time, May - June

Carolyn P. Steinhaus
13 percent time, February - April
100 percent time, May - June

Robert J. Sundstrom
54 percent time, January - April
100 percent time, May - June

3. TECHNICAL STATUS

In proposing the research activity to be conducted under the subject grant, several specific investigations were proposed for consideration during the year. Of the proposed investigations, the two focused on during the reporting period were:

(1) Reliability Analysis - Determine appropriate measures of system reliability that can be evaluated relative to some specified level of structural description, with initial emphasis on the architectural level; develop models for reliability analysis, with respect to the above

measures; and develop simulation models and computational methods for evaluating these measures. What we eventually seek are programmed reliability evaluation procedures that can be used interactively, during the process of system design, to compare the reliability of various design alternatives. Such procedures could also be used to compare the reliability of various existing systems. The evaluation procedures should be general enough to accommodate new schemes for reliability enhancement, in addition to well-known techniques. This is to be contrasted with CARE [1], for example, which was designed specifically for the evaluation of modular redundancy and standby-sparing schemes.

(2) On-line Fault Diagnosis - Determine structural and behavioral properties of systems that are conducive to their "on-line" diagnosis; investigate techniques (other than duplication) for implementing on-line diagnosis; and determine methods for altering the design of a system to improve its on-line diagnosability. As contrasted with "off-line" diagnosis, an on-line diagnostic procedure must contend with (i) system input over which it has no control and (ii) faults that occur as the system is being diagnosed. To account for these complicating factors, the study will be based on a representation of faulty digital systems as "resettable discrete-time systems," first introduced in [2].

3.1 Reliability Analysis

3.1.1 Background. A review of the current state of the art of reliability analysis reveals a situation common to relatively new fields, namely, the tendency to hold on to concepts and methodologies that were introduced when the field first began to develop. Early objects of reliability analysis were simple systems, at least in a functional sense, and simple measures could be used to determine their reliability. In a period of 30 years, however, system complexities have grown to that of a large multiprocessing computer or a complex operating system, while reliability measures have remained almost as simple as they were when applied to a radio receiver. Of course, the measures are now much more difficult to evaluate and consequently, in the area of computer reliability analysis, much of the recent research effort has focused on the derivation of formulae for calculating the values of traditional reliability measures such as "probability of success." This is not to deny the importance of "probability of success" as a measure; indeed, when the term "reliability" is narrowly interpreted it is usually given this meaning. However, in the analysis of systems with complex behavior, what constitutes "success" or "failure" can likewise be very complicated. It is this fact that is often overlooked when complex systems are analyzed using relatively simple reliability measures.

For example, a paper by Bouricious, et al. [3] presents the following formula for the reliability of a stand-by sparing configuration with N active units and S unpowered spares:

$$R = e^{-N\lambda t} \sum_{k=0}^{S} \binom{k - 1 - N\lambda/\mu}{k} C^k (1 - e^{\mu t})^k$$

where each active unit is assumed to have an exponential failure rate λ, and each spare has an exponential failure rate μ. C is the coverage. Although the failure criterion for this model was not stated, our analysis indicates that they assumed that all N active units must be functioning, and if a single active unit fails after the replacements have been exhausted, the entire system fails.

Mathur did a similar type of analysis [4] using less stringent failure criteria. He defines a hybrid system to be one which behaves as a simple NMR core after all of the spares have been depleted, so that the system fails only when there remain less than $(N + 1)/2$ unfailed modules. Mathur's equation for the reliability of this configuration is, for $S > 1$

$$R_{hybrid} = e^{-N\lambda t} e^{-S\mu t} \left[1 + \sum_{j=0}^{S-2} \binom{Nk + S}{j+1} (e^{\mu t} - 1)^{j+1} \right.$$

$$\left. + \sum_{i=0}^{N} \binom{N}{i} \binom{Nk + S}{S} \sum_{\ell=0}^{i} \frac{\binom{i}{\ell} (-1)^{i-1}}{\binom{k\ell + S}{S}} \left\{ (e^{S\mu t} e^{\ell\lambda t} - 1) - \sum_{j=0}^{S-2} \binom{k\ell + S}{j+1} \right. \right.$$

$$\left. \left. \cdot (e^{\mu t} - 1)^{j+1} \right\} \right]$$

where $k = \lambda/\mu$.

Clearly, considerable effort has gone into reliability analyses of this type. In fact, most of the combinatorial problems seem to have been solved, and in general, papers published in the last three years have essentially been restatements of previous results, with some minor modifications at best. Also, in the reliability analysis of both computer hardware and computer software, there has been a tendency in the past to blur the distinction between faults and errors, and to treat all faults identically, ignoring the fact that different classes of faults may have very different effects on system behavior. As a consequence, reliability computations that are based on such analysis methods may be quite misleading. Moreover, depending on the system being analyzed, the computations might be optimistic in one case and pessimistic in another. What is needed, then, is a more refined analysis model wherein the concepts of "fault" and "error" are distinguished, and wherein the internal "state" of a system can be accounted for when determining whether a fault causes an error.

The problems reviewed here are not presented as particular difficulties which this research effort intends to solve, but rather as examples of the type of problem which results from the fact that reliability analysis, as a field, has yet to agree on what concepts are central to the evaluation of the reliability of complex systems and, of course, how such concepts should be precisely formulated. It is the feeling of this research effort that a more comprehensive investigation of basic reliability concepts is necessary before

further meaningful work can be done at a more detailed level.

3.1.2 Computers with Faults. To establish a more refined analysis model of the type suggested above, let us begin by viewing a digital computer as a rather general type of system which, at discrete points in time, receives input data which, in turn, effects changes in the system's internal state. It will be assumed that the state set is "coordinatized" where a subset of the coordinates represent the values of those state variables that are observable as output variables. The transition structure of such a system may vary with time because faults occur or because the system is reconfigured in an attempt to recover from a fault. At a given instant of time the structure is fixed, however, and is described by a transition function which determines the state of the computing system at time i + 1, given the state at time i and the input received at time i. Formalizing this notion we have:

Definition: A (formal) computer is a system

$$C = (X, Q, \Delta)$$

where

X is a nonempty set, the input set of C,

Q is a nonempty set, the state set of C,

Δ is a sequence of functions

$$\Delta = (\delta_0, \delta_1, \delta_2, \ldots)$$

where δ_i: $Q \times X \rightarrow Q$, the transition function of C at time i ($i = 0, 1, 2, \ldots$).

Thus a computer, as defined above, is a discrete-time, time-varying system whose structure at time i is described by transition function δ_i ($i = 0, 1, 2, \ldots$). In particular, if $q \in Q$ is the state of C at time i and $a \in X$ is the input received at time i then $\delta_i(q, a)$ is the state of C at time $i + 1$. In case structure does not vary with time, that is,

$$\delta_{i+1} = \delta_i, \quad i = 0, 1, 2, \ldots \tag{3.1}$$

then C is time-invariant. Thus if $C = (X, Q, \Delta)$ is time-invariant, Δ is uniquely determined by δ_0 and C can alternatively be regarded as a (state) sequential machine with (fixed) transition function $\delta = \delta_0$.

A computer is finite-input if $|X| < \infty$ and finite-state if $|Q| < \infty$. Note that even in case a computer is both finite-input and finite-state, it is not finitely specifiable unless its structure Δ is finitely specifiable. However, in the subsequent application of this model to reliability analysis, all computers (both fault-free and faulty) of concern in the analysis will indeed be finitely specifiable.

The most general view of computer behavior is that of "string manipulation." Beginning in some initial state q_0, determined by the program to be executed and by stored data, C receives an input sequence (string) of symbols $a_0 a_1 \ldots a_{n-1}$ where $a_i \in X$ ($i = 0, 1, \ldots, n-1$) is the

input received at time i. In response to this input sequence, the computer will pass through a sequence (trajectory) of states $q_0 q_1 \cdots q_n$ where $q_i \in Q$ $(i = 0, 1, \ldots, n)$ is the state of C at time i. Thus the "state behavior" of C may be viewed as a function from the set X^* of all finite-length sequences of input symbols (including the null sequence Λ) into the set Q^+ of all finite-length sequences of states. More precisely, if $C = (X, Q, \Delta)$ and $q \in Q$, the state-behavior of C in q is a function $\alpha_q : X^* \rightarrow Q^+$ defined inductively as follows:

$$\begin{aligned} &\text{i) } \alpha_q(\Lambda) = q, \text{ for all } q \in Q, \text{ and} \\ &\text{ii) } \alpha_q(xa) = \alpha_q(x)\delta_i(q', a) \end{aligned} \tag{3.2}$$

where $i = \lg(x)$ (the length of x) and q' is the final state of the trajectory $\alpha_q(x)$, for all $q \in Q$, $x \in X^*$ and $a \in X$. It is easy to verify that this formal notion of state-behavior captures the intuitive notion discussed above. Note that α_q maps input sequences of length n into state trajectories of length $n + 1$.

Having established the concepts of "computer" and "state-behavior," we adopt a concept of "computation" that is somewhat more general than usually considered. Since computational errors may be due to faulty initial states or erroneous input symbols as well as to faulty computers, we regard computation as consisting of three things: an initial state q, an input sequence x and a state sequence y. More precisely a computation (over X and Q) is a triple (q, x, y) where $q \in Q$, $x \in X^*$ and $y \in Q^+$ such that $\lg(y) = \lg(x) + 1$. Accordingly, q, x, and y

are referred to as the initial state, input sequence and state trajectory (respectively) of the computation. Relative to a particular computer C, a computation of C is a computation of the form $(q, x, \alpha_q(x))$. The fundamental question of deciding whether a computer is behaving within specified tolerances will be based on the nature of such computations. However, even more basic than the notion of a computational error is the concept of a "fault," that is, a transient or permanent change in structure that may, in turn, cause errors.

Assuming a familiarity with the concepts of a "representation scheme" and a "system with faults" (see [5], [6]), the specification class $\mathcal{S}$ and realization class $\mathcal{R}$ that we choose in this case is the class of all computers (as defined above), that is, both $\mathcal{S}$ and $\mathcal{R}$ are equal to the class

$$\mathscr{C} = \{C \mid C \text{ is a computer}\} .$$

Moreover, we will restrict our attention to faults that occur during the use of a computer (as opposed to faults that occur during the design process) and so, in the representation scheme $(\mathscr{C}, \mathscr{C}, \rho)$, ρ is taken to be the identity function. Accordingly a computer with faults is a system

$$(C, F, \varphi)$$

where $C \in \mathscr{C}$, F is a set of potential faults of C and φ: $F \longrightarrow \mathscr{C}$ where, if $f \in F$, $\varphi(f)$ is the computer that results from fault f.

$\varphi(f)$ will alternatively be denoted C^f. One should be careful not to interpret a "fault" f as some single physical failure that occurs in the underlying system. Instead it should be interpreted as an entire sequence of physical failures that could occur during the utilization of the system. That part of a fault which changes the structure of a system at some particular instant of time, say time i, will be referred to as a "fault at time i" and will be more precisely defined in a moment.

Given the general concept of a computer with faults, let us now introduce certain restrictions that bring the concept closer to reality and result in a model that can be used for reliability analysis. If (C, F, φ) is a computer with faults it will be assumed that the fault-free specification C is time-invariant (see condition (3. 1)). This is not unreasonable since many physical systems and, in particular, most computing systems can be represented as time-invariant systems as long as there are no structural changes due to physical failures. Suppose now that a physical failure occurs where the failure may be transient, permanent, or a combination of the two, that is, a permanent physical failure that has a transient component while the permanent change is taking place. Such physical failures can be represented by (formal) faults as follows. If $C = (X, Q, \Delta)$ is a computer, a <u>fault of C</u> (at time i) is a triple (τ, π, i) where

τ: $Q \times X \rightarrow Q$, the transient component,

π: $Q \times X \rightarrow Q$, the permanent component,

i is a nonnegative integer, the time of occurrence.

The interpretation of (τ, π, i) is a physical failure that occurs between time i and time i + 1. τ is the transition function that the failing system exhibits while the failure is taking place and π is the transition function that the system exhibits after the failure has taken place. Thus, if $f = (\tau, \pi, i)$ is a fault of computer $C = (X, Q, \Delta)$, the result of f is the computer $C^f = (X, Q, \Delta^f)$ where, if $\Delta^f = (\delta_0^f, \delta_1^f, \delta_2^f, \ldots)$ then

$$\delta_j^f = \begin{cases} \delta_j \text{ if } 0 \leq j < i \\ \tau \text{ if } j = i \\ \pi \text{ if } j > i . \end{cases} \tag{3.3}$$

If, in the result of $f = (\tau, \pi, i)$, there is no permanent change in structure, that is, $\pi = \delta_{i-1}$ then f is a transient fault (at time i). A fault (π, τ, i) which represents no change whatsoever, that is, $\pi = \delta_{i-1}$ and $\tau = \delta_i$, is referred to as a null or improper fault (at time i). Finally, as discussed earlier, we want the general concept of a "fault" to include the representation of a succession of physical failures that occur during the utilization of the system. Thus, in general, a (multiple) fault of C is a sequence

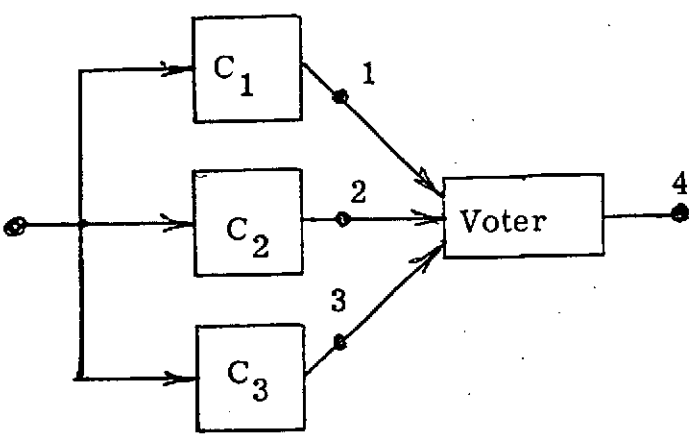
$$f = (f_{i_1}, f_{i_2}, \ldots, f_{i_k})$$

where $i_1 < i_2 < \ldots < i_k$ and f_{i_j} is a fault of C at time i_j. The corresponding result of f is an immediate generalization of definition (3.3).

To summarize, then, the computers with faults that we shall consider are of the form (C, F, φ) where C is a time-invariant computer, F is a set of faults of C (including at least one null fault), and φ: $F \rightarrow \mathscr{C}$ where $\varphi(f)$ is equal to the result of f (as defined above).

To illustrate the concepts developed above, let us consider a trivial example. (It must be emphasized that this and future examples are not intended to illustrate the full power of the formalism. They are simply given as an aid to the intuitive understanding of the definitions and results.)

Consider a TMR configuration



where each module C_i has a (fault-free) transition function $\overline{\delta}$: $\overline{Q} \times X \rightarrow \overline{Q}$ where $X = \overline{Q} = \{0, 1\}$. Then the fault-free TMR confiration is represented by a computer

$$C = (\{0, 1\}, Q, \Delta)$$

where

$$Q = \{(q_1, q_2, q_3, q_4) | q_i \in \{0, 1\}\}$$

with q_1, q_2 and q_3 representing the states of modules 1, 2 and 3 (respectively) and q_4 representing the value of the voter output. The transition structure

$$\Delta = (\delta_0, \delta_1, \delta_2, \ldots)$$

is given by a fixed function $\delta = \delta_i$ for all i, where

$$\delta((q_1, q_2, q_3, q_4), q) = (q_1', q_2', q_3', \mu(q_1', q_2', q_3'))$$

with $q_i' = \overline{\delta}(q_i, a)$ and μ equal to the majority function (realized by the voter).

Suppose now that at time 2 there is a transient struck-at-one failure at the output of module 1 and at time 4 there is a permanent stuck-at-zero failure at the output of module 3. Then this succession of failures is represented by the (multiple) fault

$$f = (f_2, f_4)$$

where f_2 is the fault at time 2 and f_4 is the fault at time 4. More specifically, f_2 is the fault

$$(\tau_2, \pi_2, 2)$$

where (letting $q_i' = \overline{\delta}(q_i, a)$):

$$\tau_2((q_1, q_2, q_3, q_4), a) = (1, q_2', q_3', \mu(1, q_2', q_3'))$$

and

$$\pi_2 = \delta .$$

f_4 is the fault

$$(\tau_4, \pi_4, 4)$$

where:

$$\tau_4((q_1, q_2, q_3, q_4), a) = (q_1', q_2', 0, \mu(q_1', q_2', 0))$$

and

$$\pi_4 = \tau_4 .$$

The result of the fault f is the computer $C^f = (\{0, 1\}, Q, \Delta^f)$

where

$$\delta_j^f = \begin{cases} \delta \text{ if } 0 \leq j < 2 \\ \tau_2 \text{ if } j = 2 \\ \pi_2 \text{ if } j = 3 \\ \tau_4 \text{ if } j = 4 \\ \pi_4 \text{ if } j > 4 . \end{cases}$$

3.1.3 Tolerance Relations and Erroneous Computations. Let us now consider the effects of faults on computer behavior, i.e., the computational errors that may be caused by faults. Recall that, in general, a computation (over X and Q) is a triple (q, x, y) where q is the initial state, x is the input sequence and y is the state trajectory of the computation. What we seek is a basis for comparing computations to determine whether an actual computation (q, x, y) is within "tolerance" of the desired computation (q', x', y'). More formally, if U is the class of all computations (over X and Q), a tolerance relation (for computations) is a relation T on the set U such that T is reflexive. If $(u, u') \in T$ we will write uTu' with the interpretation that, from the user's point of view, actual computation u is within tolerance of desired computation u'. The reflexive condition says simply that every computation is within tolerance of itself, which is certainly a reasonable requirement. Accordingly the strongest tolerance relation is the relation of equality; the weakest is the relation $T = U \times U$ where every computation is within tolerance of every other computation. The latter says that anything the computer does is acceptable and therefore represents a theoretical extreme as opposed to a practical one.

In specifying a tolerance relation, one is able to specify tolerable changes in initial state or tolerable changes in input as well as tolerable changes in state trajectory. However, if a system is assumed

to be free of initialization and input faults, it is convenient to consider tolerance relations T that only permit changes in the state trajectory of a computation. More precisely, if u, u' ϵ U where u = (q, x, y) and u' = (q', x', y') then T satisfies the condition:

$$uTu' \text{ implies } q = q' \text{ and } x = x' \tag{3.4}$$

If a tolerance relation T is so restricted, it follows that whenever u is <u>not</u> in tolerance of u', the state trajectories of u and u' must differ. During the current reporting period we have confined our attention to tolerance relations of this type and it will be assumed that condition (3.4) is satisfied, unless otherwise qualified.

Suppose now that (C, F, φ) is a computer with faults and a specified tolerance relation is being used to determine the computational integrity of computers that result from faults. In particular, suppose f ϵ F and u is a computation of the faulty computer C^f, that is, for some q ϵ Q and x ϵ X*,

$$u = (q, x, \alpha_q^f(x))$$

(α_q^f is the state-behavior of C^f in q; see (3.2).) Since the desired computation is the computation performed by the fault-free computer (for the same q and x), that is, the computation

$$u' = (q, x, \alpha_q(x))$$

it is reasonable to regard u as "erroneous" if u is not within tolerance of u'. More precisely, if T is a tolerance relation and f ∈ F we have:

Definition: A computation u of C^f is T-erroneous if $u \not T (q, x, \alpha_q(x))$ where q and x are the initial state and input sequence of u. Otherwise u is T-error-free.

When the tolerance relation is understood, we will drop the reference to T and refer to u as simply "erroneous" or, in the opposite case, "error-free." We will also say that a T-erroneous computation of C^f is caused by f. Finally, if f can cause no T-erroneous computations then f is T-tolerated.

It should be noted that the concept of a T-erroneous computation can capture the notion of an internal error as well as an input-output error. To illustrate, consider the TMR example used earlier and suppose T is the relation of equality (identity) on U, that is

$$uTu' \text{ if } u = u'.$$

Then the fault $f = (f_2, f_4)$, considered in the earlier example, can cause T-erroneous computations even though f cannot cause input-output errors (assuming the modules are properly initialized). To be more specific, let us suppose the module transition function $\overline{\delta}$ is given by:

(q, a)	$\overline{\delta}$(q, a)
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	0 .

Then, for example, if q = (0, 0, 0, 0) and x = 101 then

$$\alpha_q^f(x) = (0,0,0,0)(1,1,1,1)(1,1,1,1)(1,0,0,0)$$

since the transition function at time 2 is τ_2. On the other hand

$$\alpha_q(x) = (0,0,0,0)(1,1,1,1)(1,1,1,1)(0,0,0,0).$$

Thus the computations u = (q, x, $\alpha_q^f(x)$) and u' = (q, x, $\alpha_q(x)$) are not equal, that is, u$\not T$u' and hence u is a T-erroneous computation.

To continue the example, suppose T' is a second tolerance relation which requires only that values on the output line (coordinate 4) be what they should be. More precisely, (q, x, y)T'(q, x, y') if y and y' have the same length, say n, and the i^{th} state of y has the same 4^{th} coordinate as the i^{th} state of y', i = 0, 1, . . . , n-1. Given that T' is the tolerance relation of interest, it can be shown that the fault $f = (f_2, f_4)$ does not cause any T'-erroneous computations (provided all module states are the same when the computation begins). In other words, although f can cause internal errors (according to tolerance relation T), it can cause no input-output errors (according

to tolerance relation T').

In general, any analysis of computing system reliability must be based on some underlying criteria that defines "acceptable" or "tolerable" behavior. Since a tolerance relation is simply a formal statement of such criteria, it too is fundamental to reliability analysis. In many cases, the tolerance relation that underlies the analysis is not stated explicitly but is nevertheless easily inferred. In other cases, however, it is extremely difficult to judge what the analyst regards as tolerable behavior and, consequently, the results of the analysis are difficult to interpret.

When the underlying tolerance relation (or some equivalent thereof) is not stated explicitly, results of the analysis can also be misleading. For example, in a paper by Mathur ([7], 1971), the problem of optimally allocating 7 identical modules is considered with the conclusion that a standby replacement system (i.e., 1 active unit and 6 unpowered spares) is "...clearly... superior to hybrid systems..." (i.e., (3,4) or (5,2), with voter). What is ignored is the fact that different tolerance relations are used to calculate the respective reliabilities. With a single active unit an error yields an incorrect output, while with the hybrid system an error results in an incorrect internal state, but the output is still correct. The importance of this distinction is, of course, application dependent, but certainly the distinction must be kept in mind.

Based on a survey of reliability measures and analysis techniques that are currently being used, it is our firm belief that the appropriate starting point for reliability analysis is a precise definition of what constitutes "acceptable" or "tolerable" behavior. Moreover, the definition should be general enough to permit the specification of relatively complex tolerance criteria involving multivariable descriptions of performance, various levels of degraded performance, limits on the duration of time and/or number of times that performance is below a given level, and so forth. It is these considerations that have motivated our development up to this point, culminating in the concept of a "tolerance relation" which permits this kind of precise specification. Accordingly, the subsequent investigation of reliability analysis is based on the assumptions that the computer to be analyzed is formally described as a computer with faults and tolerable behavior is formally described by a tolerance relation defined on computations.

3.1.4 Reliability Measures. In analyzing the reliability of a computing system, one must first specify just what is meant by the term "reliability" since the word has taken on a variety of special meanings. Generally, by reliability we will mean a sequence of one or more numbers that reflect the ability to rely on a system. In particular, if the system is a computer with faults, the numbers reflect the ability to rely on the computations of the computer. Accordingly, a reliability measure is a function from systems into sequences of numbers whose value, for a particular system, is the

reliability of that system. Thus, in the context of this investigation, a reliability measure is a function from the class of all computers with faults into some Cartesian product of sets of numbers. In many cases, the Cartesian product is one-dimensional, that is, the value of the measure is a single number. If the product has more than one dimension, the measure will be referred to as multi-dimensional.

When used in the above sense, reliability has a very general meaning and includes such concepts as mean-time-to-failure, availability, recoverability, effectiveness, etc. as well as the usual strict definitions of the term that are based on some concept of successful (adequate, acceptable, tolerable) operation. Thus the term "reliability," as used above, is synonymous with the term "utility" as used in the statement of the augmented objectives (see Section 1). Unfortunately, neither term is especially well-suited for the meaning given here since reliability is often given a more restricted meaning and utility a more general meaning.

To begin our investigation of reliability measures for computers with faults, let us consider first how the usual, more restricted meaning of "reliability" translates into the terms we have developed. As defined, for example, by the Radio-Electronics-Television Association in 1955, reliability is the "probability of a device performing its purpose adequately for the period of time intended under the operating conditions encountered." Translating into our terms, this

becomes the "probability that a computer with faults behaves within tolerance for some specified utilization period." More precisely, if $C = (C, F, \varphi)$ is a computer with faults, T is a tolerance relation on computations, and $[0, t] = \{0, 1, \ldots, t\}$ is the utilization period then

Definition: The (strict) reliability R(C) of C is the probability that the computation of C in the interval $[0, t]$ is T-error-free.

The reliability measure, in this case, is the function R from computers with faults into the real interval $[0, 1]$, where R(C) is the reliability of C.

Let us now examine what suffices to compute the values of this measure. Conceptually, for each computer with faults $C = (C, F, \varphi)$, we must determine an underlying probability space $\mathscr{P}_C$ that will suffice to determine the reliability R(C). More specifically, if $\mathscr{P}_C = (S, \mathcal{E}, P)$, where S is the sample space, $\mathcal{E}$ is the event space (a σ-algebra of subsets of S) and P is the probability measure, we must determine choices of S, $\mathcal{E}$, and P that will determine R(C). Beginning with the sample space S, elements here must represent elementary outcomes, namely, computations of C. Since a computation of C is actually a computation of C^f for some $f \in F$, it suffices to let S be the set

$$S = Q \times X^t \times F \tag{3.5}$$

where $X^t = \{x \mid x \in X^* \text{ and } lg(x) = t\}$. The interpretation of a sample (q, x, f) is that the initial state is q, the input sequence is x and the (multiple) fault that occurred is f. Thus, given a sample (q, x, f) the corresponding computation (in the interval $[0, t]$) is the computation

$$(q, x, \alpha_q^f(x)).$$

The event space $\mathcal{E}$ must be chosen so that it contains the event

$$E = \{(q, x, f) \mid (q, x, \alpha_q^f(x)) \text{ is T-error-free}\} \tag{3.6}$$

and permits the definition of a probability measure $P: \mathcal{E} \rightarrow [0, 1]$ (with the usual interpretation that, for all $D \in \mathcal{E}$, $P(D)$ is the probability that the outcome is in the event D). It follows then that $\mathcal{P}_C$ suffices to determine R(C) since, if E is the event given above, then

$$R(C) = P(E) \tag{3.7}$$

(Technically speaking, the above equation should be regarded as the definition of R(C), for it is here that we give precise meaning to the word "probability.")

Upon closer examination of this reliability measure, the reason for the careful development of the preceding sections should now be clear. We note first that the underlying probability measure P is

defined on initial states, input sequences and faults, and not simply faults alone. This permits initial-state-dependent or data-dependent performance to be accounted for in the reliability analysis. Most conventional approaches, on the other hand, account only for the probabilistic nature of faults, thereby ignoring effects of initialization and data.

Second, we note that what constitutes "success" (as the term is usually employed in strict definitions of reliability) is precisely specified by a tolerance relation T. Moreover, the tolerance relation is not restricted to instantaneous "snapshots" of structure or behavior but, instead, is defined on complete computations. This permits the past history of the computation and, in particular, the present state of the computer to be accounted for in the judgment as to whether performance is successful.

To illustrate these remarks, let us suppose the "computer" to be analyzed is a simple two-state device, namely, a trigger flip-flop (alternately referred to as a T-flip-flop or mod-2 counter). The fault-free representation of this device is the computer $C = (\{0, 1\}, \{0, 1\}, \delta)$ where δ is given by the table:

(q, a)	δ(q, a)
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	0

that is, C stays in the same state if the input is 0 and changes state if the input is 1. Suppose further that the only device failures having an appreciable probability of occurrence are stuck-at failures at the output where the hazard rate (failure rate) is a constant λ. Then, by the usual method of reliability analysis (see [8], for example), the reliability $\underline{R}(C)$ is given by the equation:

$$\underline{R}(C) = e^{-\lambda t} \tag{3.8}$$

where $[0, t]$ is the utilization period. (The notation $\underline{R}$ is used to distinguish this measure from the measure R defined above.) What is being assumed here is that the flip-flop fails as soon as a stuck-at failure occurs. This certainly simplifies the measure but, at the same time, ignores the effects of initial state, input, the transition function and, most importantly, what the user regards as tolerable behavior.

Let us now examine how our more refined measure R can account for effects ignored by $\underline{R}$. In particular, let us suppose that the user is interested in the value of the output only when the computation terminates. In other words, the tolerance relation is the relation T where $(q, x, y)T(q, x, y')$ if the final state of trajectory y is equal to the final state of trajectory y'. If, further, we let F be the set of all faults of C that represent (permanent) stuck-at failures and $f \in F$, it follows that the computation $(q, x, \alpha_q^f(x))$ is T-error-free if and only if the final state of $\alpha_q^f(x)$ is equal to the final state of $\alpha_q(x)$. Accordingly

the event E, corresponding to all error-free computations (see (3.6)), is the event

$$E = \{(q, x, f) \mid \text{Final state of } \alpha_q^f(x) = \text{Final state of } \alpha_q(x)\} .$$

It remains then to determine the probability of E which, by (3.7) gives the reliability R(C). This is easily done if we consider the following events (subsets of $\{0, 1\} \times \{0, 1\}^t \times F$):

$$N = \{(q, x, f) \mid f \text{ is null or } f = (\tau, \pi, i) \text{ where } i \geq t\}$$

$$A_0 = \{(q, x, f) \mid q = 0\}$$

$$A_1 = \{(q, x, f) \mid q = 1\}$$

$$B_0 = \{(q, x, f) \mid \text{Parity}(x) = 0\}$$

$$B_1 = \{(q, x, f) \mid \text{Parity}(x) = 1\}$$

$$C_0 = \{(q, x, f) \mid f = (\tau, \pi, i) \text{ where } i < t \text{ and } \tau = \pi = \sigma_0\}$$

$$C_1 = \{(q, x, f) \mid f = (\tau, \pi, i) \text{ where } i < t \text{ and } \tau = \pi = \sigma_1\} .$$

(σ_0 and σ_1 denote the functions "constant 0" and "constant 1.")
These events can be paraphrased as follows:

N: No fault occurs in the interval $[0, t)$.

A_i: The initial state is i.

B_i: The number of 1's in the input sequence is equal to i (mod 2).

C_i: A stuck-at-i fault occurs in the interval $[0, t)$.

We note first that $N \subseteq E$ since, here, no fault occurs before the end of the utilization period. To determine other events in E, consider, for example, the event $D = A_0 B_1 C_1$ (the intersection of these three events). If $(q, x, f) \in D$, since f is a stuck-at-1 fault that occurs during the utilization period, the final state of $\alpha_q^f(x)$ is equal to 1. But q = 0 and

and the parity of x is odd, so the final state of $\alpha_q(x)$ is also equal to 1. In other words, such computations are T-error-free and we conclude that $D \subseteq E$. By similar reasoning for other compound events, it can be established that:

$$E = N \cup A_0B_0C_0 \cup A_1B_1C_0 \cup A_1B_0C_1 \cup A_0B_1C_1 .$$

Since the events on the right-hand side are mutually exclusive and assuming the events A_i, B_i and C_i are independent (a reasonable assumption in this case) we have:

$$P(E) = P(N) + [P(A_0)P(B_0) + P(A_1)P(B_1)]P(C_0)$$

$$+ [P(A_1)P(B_0) + P(A_0)P(B_1)]P(C_1) .$$

Here, under the earlier assumption regarding failure rates,

$$P(N) = e^{-\lambda t}$$

and, since $C_0 \cup C_1 = \overline{N}$,

$$P(C_0) + P(C_1) = 1 - e^{-\lambda t} .$$

If we assume further that all sequences in $\{0, 1\}^t$ are equally likely and the initial state is always 0 then

$$P(A_0) = 1 \qquad P(B_0) = 0.5$$

$$P(A_1) = 0 \qquad P(B_1) = 0.5 .$$

Substituting these values in the above expression, we obtain the reliability of C, that is,

$$R(C) = P(E) = e^{-\lambda t} + 0.5(1 - e^{-\lambda t}) . \qquad (3.9)$$

Comparing (3.9) with (3.8), we see that the reliability of the flip-flop is greater when error-free behavior is judged according to the tolerance relation T. Thus, for example, if the failure rate λ is 10^{-4} failures/hour and the utilization period is 1000 hours then, according to (3.8):

$$\underline{R}(C) = e^{-0.1} = 0.90 .$$

On the other hand, applying (3.9):

$$R(C) = e^{-0.1} + 0.5(1 - e^{-0.1}) = 0.95 .$$

If some other set of assumptions were made regarding the probabilistic nature of C (i.e., the probability measure P), the extent of the improvement might differ but, in no case, would R(C) be less than $\underline{R}$(C).

3.1.5 Topics for Further Investigation. The ability to define a reliability measure and apply it in the manner illustrated above demonstrates both the feasibility and the potential of this approach to reliability analysis. Due to the generality of the framework, there are relatively few limitations on the types of systems, faults, tolerance relations, and reliability measures that are describable within the formalism. However, to say that something is describable (in the

sense that a description exists) is usually not enough; it must also be the case that an object remains describable when the description process is subject to constraints on money, manpower, time and space. If computers are used in the description process (which will be mandatory in many cases), time includes computer run-time and space includes computer storage. Therefore, in parallel with our further investigation of appropriate tolerance relations and reliability measures we intend to investigate various means of simplifying their description. If the price paid for simplification is a less accurate description, the effects of such inaccuracies will also be investigated.

These remarks regarding the economics of the description process apply as well to the process of evaluating the reliability of a computer according to some reliability measure, given that the computer (with faults) and the measure have already been described. One possible means of simplifying a complex evaluation process would be to decompose it via a decomposition of the measure. In other words, attempt to define submeasures which can be more easily evaluated and, in turn, combined to yield the value of the measure. We believe this approach deserves investigation and we intend to actively pursue it during the next reporting period. In cases where the calculation of exact values is infeasible, due to the computational complexity of the algorithm or to insufficient knowledge of the underlying probability measure, methods of calculating approximate values will be investigated.

If the measure is decomposed, as suggested above, such methods could also be used to calculate the values of submeasures.

Aside from these questions regarding how measures are described and how they are evaluated, a more fundamental question is the determination of just what should be measured. The strict reliability measure discussed in the previous section (or, more properly, the class of strict reliability measures, since the tolerance relation T can be varied) is but one measure among many that could be applied. In viewing what others have accomplished in this regard, we believe that "recoverability," or what is usually referred to as "coverage" [9], deserves much more study with regard to its measure. This includes all the aspects we have discussed for reliability measures in general, that is, a precise definition of what is meant by recoverability (as defined on computers with faults), economic descriptions of whatever objects the recoverability measure is based on, and an efficient means of evaluating the recoverability of a given computer.

In support of the need for such an investigation, Bouricius, et al. [9] state that "...coverage... is the single most important parameter in high-reliability system design. Changing the coverage from 1 to about 0.98 can result in orders of magnitude degredation in system mission time." Also, we have programmed several reliability functions involving a coverage parameter, and have observed, through interactive

terminal sessions, that coverage is indeed a critical parameter. Moreover, the relative importance of coverage increases with increasing system reliability.

As a consequence of this assessment, just prior to the end of the present reporting period we initiated an investigation of recoverability measures and their evaluation. The initial step has been to explore just what is meant by "recovery" and "time of recovery" in the context of a computer with faults and relative to some specified tolerance relation. Once these questions are settled, precise formulation of a recovery measure will begin and the investigation will proceed as outlined above.

3.2 On-line Diagnosis

3.2.1 Background. In many applications, especially those in which a computer is being used to control some process in real-time, (e.g., telephone switching, flight control of an aircraft or spacecraft, etc.) it is desirable to constantly monitor the performance of the system, as it is being used, to determine whether the actual system is within tolerance of the intended system. Informally, by "on-line diagnosis" we mean a monitoring process of this type where the extent of the diagnosis depends on the meaning of "within tolerance." Thus, for example, if being within tolerance means having the same input-output behavior, then on-line diagnosis becomes on-line "detection." In the special case where the implementation of on-line

diagnosis is completely internal to the system being diagnosed, it is referred to as "self diagnosis" or "self checking."

The incorporation of special hardware for the purpose of on-line diagnosis dates way back to the relay computers developed by Bell Laboratories in the early-to-mid 1940's, where biquinary codes were used to dynamically check the operation of the computer [10]. A more general look at codes for checking logical operations was first taken by Peterson and Rabin in 1959 [11] where they showed that combinational circuits can vary greatly in their inherent on-line diagnosability. The use of coding techniques in the design of self-checking circuits was further explored by Carter and Schneider in 1968 [12] and by Anderson in 1971 [13]. In addition, a number of special on-line diagnosis methods have been considered which apply to specific hardware subsystems such as adders, counters, etc. (see [14], for example).

A theoretical study of on-line fault diagnosis was initiated under NASA Grant NGR23-005-463. The motivation for this study was the increasing use of computers in real-time applications where (i) erroneous operation can result in the loss of human life and/or large sums of money and (ii) interruptions in the operation, for the purpose of off-line diagnosis, are intolerable. In particular, our discussions with NASA-Langley regarding such applications were influential in precipitating this study.

The initial problem considered in our study was the formulation of an appropriate class of system models (i.e., a class of "systems with faults") that could serve as a basis for the theoretical study of on-line diagnosis. This effort was motivated by the observations that (i) conventional models of time-invariant systems (e.g., sequential machines) are inadequate since they cannot represent the dynamics of a system as faults occur and (ii) many systems are originally designed with an explicit reset mechanism (e.g., the clear button on an adding machine) or must have a reset capability due to their intended implementation. These observations led to the formulation of a class of resettable discrete-time systems which adequately represent the structure and behavior of both "fault-free" and "faulty" systems in an on-line diagnosis environment. Given a (resettable, discrete-time) system S, a fault f of S is represented by a triple $f = (S', \tau, \theta)$ with the interpretation that S is transformed into system S' at time τ with transient state behavior θ. The result of f is taken to be the system S^f which looks like S up to time τ and like S' thereafter.

Once such systems were defined, the next problem considered was the formulation of notions of fault tolerance, error, diagnosability, realization, etc. that have a meaningful interpretation in the context of on-line diagnosis. To summarize briefly, if S is a system and f is a fault of S, we say that f is tolerated if the resulting faulty system

S^f is able to mimic some desired behavior as specified by a reduced system $\tilde{S}$. Otherwise, f causes errors (i.e., erroneous outputs) for some initial conditions and input sequences. Our notion of on-line diagnosis involves an external detector D (assumed to be fault-free) and a maximum delay k within which any error caused by a fault must be detected. More specifically, a system S with faults F is (D, k)-diagnosable if, for all f in F,

(i) D responds negatively if S is fault-free,

and

(ii) D responds positively within k time steps of the first occurrence of an error caused by f.

After the above concepts were made precise, certain fundamental questions were posed and their investigation was initiated. The research outlined above was first described in the technical report "On-line Diagnosis of Sequential Systems" [15].

3.2.2 Recent Activity. During this reporting period we have continued our investigation of on-line diagnosis and we have obtained results which have substantially increased our knowledge of the subject. The activity during this period has focused on the diagnosis of two sets of faults; namely, the set of "unrestricted faults" and the set of "unrestricted component faults."

The set U of unrestricted faults of a system is defined to be simply the set of all faults of that system. Aside from representing

a "worst-case" fault environment, there are certain practical reasons for considering U, at least at the outset. In particular, as the scale of integrated circuit technology becomes larger, it becomes more difficult to postulate a suitably restricted class of faults such as the class of all "stuck-at" faults. Moreover, although other failure models such as bridging failures have been proposed and studied (see [16] and [17] for example), little is known about the diagnosis of such failures. In addition, intermittent and multiple failures are also possible and are even more difficult to model. Finally, for a given failure it may be impossible to determine the θ function of the fault caused by this failure. Thus fault sets which do not restrict the transient state behavior θ are advantageous.

Given the background of techniques that have been proposed and, in many cases, used to improve the on-line diagnosability of a system, the following question arises quite naturally. With regard to any technique that might be employed, how complex must the diagnosing system be as compared to the system being diagnosed, if the latter is to be on-line diagnosable for some prescribed set of faults? To answer this question, one must, of course, designate the complexity measure. As a measure of system complexity, we have chosen the number of reachable internal states. This measure reflects the memory capacity of a system and, without further restrictions on system structure, it's the only measure of structural complexity that has a reason-

able interpretation. Here we have shown that if a system is on-line diagnosable for the unrestricted set of faults then the detector is at least as complex as the specification. Moreover, this result holds even when the allowed time delay for error detection is arbitrarily large.

One means of diagnosing the unrestricted faults of a system is to use a detector which consists of a duplicate of the system being diagnosed and a matching circuit which can dynamically compare the operation of the system with its duplicate. For systems which have (delayed) inverses, that is, systems which are information lossless, an alternative means of performing unrestricted fault diagnosis is the use of a loop check. Our research here has established that an inverse system can always be used for on-line unrestricted diagnosis if it too is information lossless. Although the lossless condition is sufficient, it is shown further that there exist systems for which a lossy inverse can also be used for on-line unrestricted fault diagnosis.

Since not every system has an inverse, let alone one which can be used for unrestricted fault diagnosis, it is not always possible to apply this technique directly. However, we have shown that every system has a realization to which this scheme can be successfully applied.

A detailed discussion of the above results has been documented in a paper entitled "On-line Diagnosis of Unrestricted Faults" [18] which has been submitted for publication to the IEEE Transactions on Computers.

The on-line diagnosis of systems, which are structurally decomposed and are represented as a network of smaller systems, has also been investigated. The fault set considered here is the set of unrestricted component faults; namely, the set of faults which only affect one component system of the network. A characterization of networks which can be diagnosed using a combinational detector has been obtained and it is shown that any network can be made diagnosable in the above sense by the addition of one component with a complexity as great as the most complex component in the network. In addition, a lower bound has been obtained on the complexity of any component, the addition of which is sufficient to make a particular network combinationally diagnosable.

A detailed discussion of all of the work to date on on-line diagnosis has recently been documented in the technical report "On-line Diagnosis of Sequential Systems: II" [19]. This report includes modifications of material covered in an earlier report ("On-line Diagnosis of Sequential Systems" [15]), and rigorously establishes the results reviewed above.

3.2.3 Topics for Further Investigation. Although much progress has been made towards achieving a thorough understanding of on-line diagnosis, many possibilities for further investigation remain. Except for research on the diagnosis of networks of systems, our investigation has been dealing with totally unstructured systems. Such an

approach is well suited to the development of formal concepts involved in the theory of on-line diagnosis. It is also well suited to the investigation of these concepts, provided that the faults in question do not depend on a more refined knowledge of structure as, for example, when the faults are unrestricted. On the other hand, many interesting and important questions are better studied in a more structured environment. One reason for this is that, with a structured system, we can consider the causes of faults. For example, given an unstructured system it makes no sense to speak of the set of faults caused by component failures of a certain type or by bridging failures. However, given a structured representation of a system (e.g., a circuit diagram) we can discuss these and other types of failures (causes) and determine the resulting faults (effects).

There are many different structural levels that could prove useful to a further investigation into the theory of on-line diagnosis. Two levels which we believe will be important are: the binary state-assigned level and the logical circuit level. These levels and the basis for their potential usefulness are explained below.

A machine M is said to be binary state-assigned if $Q = \{0, 1\}^n$ for some positive integer n. Given such a machine we can speak of stuck-at-0 and stuck-at-1 and other types of memory failure. The faults corresponding to these failures can be enumerated and comparisons can be made between various schemes for diagnosing these faults.

Memory faults have been studied before in other contexts (see [20] and [21] for example) and they are an important class of faults for a number of reasons. As we have seen, only a limited amount of structure is needed to discuss them. Thus memory faults can be analyzed before the circuit design of the machine is complete. Also, it is memory which distinguishes truly sequential systems from purely combinational (one-state) systems. Combinational systems are inherently easier than sequential systems to analyze and a number of techniques for the on-line diagnosis of such systems are known (see [14] and [22] for example).

A system possesses structure at the logical circuit level if a representation of the system is given in terms of a logical circuit composed of primitive logical elements. These may be of the AND-OR variety, threshold elements, or any similar elements of a "building block" nature depending upon the technology being considered. This level is useful for investigating failures in the primitive components.

Further work could also be performed at the network level of structural detail. At this level one could study the problem of implementing on-line diagnosis on a whole computer whereas with the other levels the emphasis would be on diagnosing one module. Note that in our definition of diagnosis the detector is not constrained to give simply a yes-no response. It could also provide extra information for use in automatic fault location. Thus at this level the problem of which

subsystems must be explicitly observed by the detector to achieve some desired fault location property could be studied.

One problem that requires extension of our present model (at any structural level) is the problem of automatic reconfiguration of the system under the control of the detector. To study this problem, the model used would have to allow for feedback from the detector to the system it is observing. The question of how such an extension should be made is an intersting one and, if answered satisfactorily, the resulting model could serve as a basis for a systematic investigation of reconfiguration techniques.

REFERENCES

[1] F. P. Mathur, "Reliability Estimation Procedures and CARE: The Computer-Aided Reliability Estimation Program," JPL Quart. Tech. Rev., vol. 1, October 1971.

[2] J. F. Meyer, "Semiannual Status Report No. 3; Theory of Reliable Systems," NASA Grant NGR23-005-463, January 1973.

[3] W. G. Bouricius, W. C. Carter, D. C. Jessep, P. R. Schneider and A. B. Wadia, "Reliability Modeling for Fault-Tolerant Computers," IEEE Trans. on Computers, vol. C-20, No. 11, November 1971, pp. 1306-1311.

[4] F. P. Mathur and A. Avizienis, "Reliability Analysis and Architecture of a Hybrid-Redundant Digital System: Generalized Triple Modular Redundancy with Self-Repair," AFIPS Conf. Proc., (Sprint Joint Computer Conference), vol. 36, May 1970, pp. 375-383.

[5] J. F. Meyer, "Sequential Behavior and its Inherent Tolerance to Memory Faults," Proc. of 5th Hawaii Int. Conf. on System Sciences, January 1972, pp. 476-478.

[6] J. F. Meyer, "Theory of Reliable Systems," Systems Engineering Laboratory Tech. Rpt. No. 73, The University of Michigan, Ann Arbor, July 1973.

[7] F. P. Mathur, "On Reliability Modeling and Analysis of Ultra-reliable Fault Tolerant Digital Systems," IEEE Trans. on Computers, vol. C-20, No. 11, November 1971, pp. 1376-1382.

[8] M. L. Shooman, Probabilistic Reliability: An Engineering Approach, McGraw Hill Book Co., Inc., New York, 1968.

[9] W. G. Bouricius, W. C. Carter, and P. R. Schneider, "Reliability Modeling Techniques for Self-Repairing Computer Systems," Proc. ACM 1969 Ann. Conf., pp. 295-309.

[10] S. B. Williams, "Bell Telephone Laboraties Relay Computing System," Ann. Computation Lab., Harvard University, vol. XVI, 1948, pp. 41-53.

[11] W. W. Peterson and M. O. Rabin, "On Codes for Checking Logical Operations," IBM Journal, vol. 3, April 1959, pp. 163-168.

[12] W. C. Carter and P. R. Schneider, "Design of Dynamically Checked Computers," Proc. of the IFIPS, Edinburgh, Scotland, August 1968, pp. 878-883.

[13] D. A. Anderson, "Design of Self-checking Digital Networks using Coding Techniques," Coordinated Science Laboratory Rpt. R527, The University of Illinois, Urbana, September 1971.

[14] F. F. Sellers, M. Hsiao and L. W. Bearnson, Error Detection Logic for Digital Computers, McGraw-Hill Book Co., Inc., New York, 1968.

[15] R. J. Sundstrom, "On-line Diagnosis of Sequential Systems," Systems Engineering Laboratory Tech. Rpt. No. 72, The University of Michigan, Ann Arbor, July 1973.

[16] K. C. Y. Mei, "Bridging and Stuck-at Faults," in Dig. 1973 Int. Symp. Fault-Tolerant Computing, June 1973, pp. 91-94.

[17] A. D. Friedman, "Diagnosis of Short Faults in Combinational Circuits," in Dig. 1973 Int. Symp. Fault-Tolerant Computing, June 1973, pp. 95-99.

[18] J. F. Meyer and R. J. Sundstrom, "On-Line Diagnosis of Unrestricted Faults," (submitted for publication, IEEE Trans. on Computers).

[19] R. J. Sundstrom, "On-line Diagnosis of Sequential Systems: II," Systems Engineering Laboratory Tech. Rpt. No. 81, The University of Michigan, Ann Arbor, July 1974.

[20] J. F. Meyer, "Fault Tolerant Sequential Machines," IEEE Trans. on Computers, vol. C-20, No. 10, October 1971, pp. 1167-1177.

[21] K. Yeh, "A Theoretic Study of Fault Detection Problems in Sequential Systems," Systems Engineering Laboratory Tech. Rpt. No. 64, The University of Michigan, Ann Arbor, June 1972.

[22] W. H. Kautz, "Automatic Fault Detection in Combinational Switching Networks," Stanford Research Institute Project No. 3196, Tech. Rpt. No. 1, Menlo Park, California, April 1961.